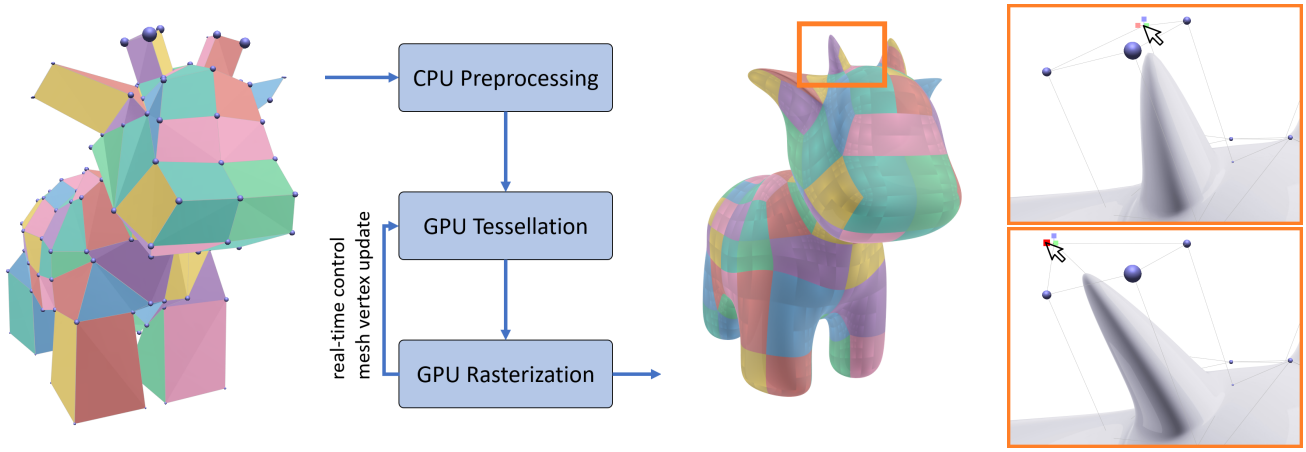
Vision, Modeling, and Visualization (2021)
B. Andres, M. Campen, and M. Sedlmair (Eds.)

Real-Time Gaussian-Product Subdivision on the GPU

Alexander Komar and Reinhold Preiner

Institute of Computer Graphics and Knowledge Visualization, Graz University of Technology, Austria



(a) *Input covariance mesh* **(b)** *Pipeline* **(c)** *Rasterized non-linear limit surface* **(d)** *Real-time response*

Figure 1: *Example covariance mesh model (a), where Gaussian covariances attached to the control mesh vertices are visualized as blue ellipsoids. This mesh defines a non-linear Gaussian-Product subdivision surface (c), rendered as tessellated parametric patches in real-time by our method (shading indicates patch tessellation and parametrization). Our work presents a GPU pipeline (b) that is the first to render such probabilistic subdivision surfaces in real-time, enabling instant visual feedback on dynamic covariance mesh updates, for instance, in an interactive modeling scenario for designing and moving sharp features (d).*

Abstract

We propose a real-time technique for rendering Gaussian-Product subdivision surfaces. This is achieved by our real-time subdivision pipeline, able to accept the base of Gaussian-Product subdivision, that is, a covariance mesh, which extends regular base meshes by storing additional 3x3 covariance matrices per vertex. Our technique evaluates the non-linear limit subdivision surface by computing B-spline patches embedded in a 9-dimensional dual space, where the subdivision scheme becomes linear. We construct and evaluate these B-spline patches using real-time tessellation capabilities of current GPUs. We analyzed the performance of our technique on all supported subdivision levels, and provide an analysis of its visual quality and geometric accuracy.

1. Introduction

Subdivision surfaces are a widely used technique to create smooth surfaces from very coarse control geometry, enabling a wide range of applications in geometric modeling, computer games, and offline rendering in movie productions. The base concept is straightforward but powerful. A coarse control mesh gets refined by dividing each face into smaller faces until a limit surface is reached. Among the various different subdivision schemes developed since the late 1970s, one of the earliest but still most popular ones is the method of Catmull and Clark [CC78], which is primarily applicable to quadrilateral control meshes and thus often used for high-performance and offline rendering in movie production. Here, each quad face gets subdivided into 4 smaller quad faces, thereby introducing 5 new vertices in the refinement step. This scheme allows for simple and versatile geometry definitions, allows designers to model shapes based on intuitive quadrilateral control meshes, and enables a straightforward procedural generation for their smooth

DOI: 10.2312/vmv.20211368

limit surfaces, which exhibit the often desired C^2 continuity everywhere, except at extraordinary vertices where they are C^1 continuous.

One of the major drawbacks of this approach is its lack of native support for the specification and generation of sharp features. To this end, several extensions to the Catmull-Clark scheme were proposed, the most widely used one depending on an explicit tagging of control mesh edges as sharp or semi-sharp creases [DKT98]. Another approach widely used in practical modeling is the explicit insertion of additional (narrow) edge loops, which allows sharper features to be natively generated by standard subdivision schemes, but also increases the granularity of the control mesh around the intended features and thus leads to larger and more complex control meshes. In both these techniques the introduction of geometric features, in one or the other way, requires a manual encoding in – and thus a particular awareness of – the control mesh topology, rather than the geometric vertex configuration alone.

Recently, an alternative approach was proposed that allows to model features by encoding enriched surface properties directly at the vertices of a control mesh [PBW19]. Here, vertex data is extended by including an anisotropic Gaussian distribution of possible surface locations around each vertex, as illustrated by the blue ellipsoids in Figure 1 (a). Using this extra information, sharp or semi-sharp features can be constructed without increasing the granularity of the base mesh, using a non-linear, probabilistic subdivision based on Gaussian products. An example is given by the sharpened horn seen in Figure 1 (d). This *Gaussian-Product Subdivision* (GPS) scheme offers new possibilities in interactive surface design by focusing on a vertex-based specification of the surrounding surface rather than a topology-based encoding.

To utilize these potentials in practical real-time applications, such as real-time animation or interactive modeling with real-time feedback, a basic prerequisite is the ability to generate and render the limit subdivision surface defined by the current control mesh on the fly. For classic subdivision schemes, this task was mainly enabled by recent advances in the performance and capabilities of consumer-grade GPUs, but also benefited from earlier insights into analytic properties of the limit surfaces to be generated.

In this paper, we present a novel GPU rendering pipeline that allows to generate these probabilistic subdivision surfaces on the fly. We realized the non-linear subdivision by a parametric evaluation of individual limit surface patches, utilizing the tessellation capabilities of modern GPUs [BFK*16]. The major challenge for this evaluation on Gaussian control meshes is the fact that the involved basis functions do not only incorporate 3D positions, but additional covariance matrices, resulting in 9-dimensional B-spline patches to be evaluated on the fly. To perform this task efficiently on the GPU, we exploit a formal relationship between this non-linear probabilistic subdivision scheme and standard linear subdivision, which provide a parametric definition of its limit surface [Sta98b]. This results in a novel efficient pipeline enabling real-time rendering and high-quality shading of probabilistic subdivision surfaces.

We provide a thorough description of an implementation in OpenGL, and analyze the rendering performance of our pipeline as well as its geometric accuracy and surface quality at a variety of test models. Based on this approach, we envision novel applications in rendering and modeling based on this versatile subdivision scheme.

2. Background and Related Work

Our technique directly relates to formal foundations of subdivision surfaces, probabilistic Gaussian-Product subdivision, and GPU-based real-time rendering approaches.

2.1. Subdivision Surfaces

Subdivision surfaces were first introduced by Catmull and Clark [CC78] and Doo and Sabin [DS78], independently in 1978. The Catmull-Clark algorithm is a generalization of bi-cubic B-splines to arbitrary quadrilateral patches. Generalizations of parametric patches to other domains were also developed at the same time [DS78] and later on [Loo87, Kob00]. Extensions to the Catmull-Clark algorithm were later proposed to describe sharp or semi-sharp features [DKT98] or to prescribe local surface normals [BLZ00]. The former was used in several animated full-feature films due to its flexibility and extensive freedom in design. Several advanced approximating and interpolating, as well as linear and non-linear subdivision schemes have been introduced later on, whose detailed discussions lie beyond the scope of this paper. For a further reference on the vast zoo of subdivision techniques, we refer the reader to the survey of Cashman [Cas12].

2.2. Gaussian-Product Subdivision

Gaussian-Product Subdivision (GPS) [PBW19] represents an alternative, probabilistic subdivision approach that can be directly attached to a point-based reconstruction process based on Gaussian mixture distributions [PBW19]. The method builds on an extended mesh model, a *covariance mesh*, in which each vertex additionally stores an anisotropic 3×3 covariance matrix Σ that describes the probabilistic alignment of the surface around its mean vertex μ. This information offers more freedom in design, e.g. to model sharp features along the intersection of two Gaussian distributions.

Including the covariance data in the subdivision process leads to a probabilistic subdivision scheme based on non-linear combinations: two Gaussian distributions Θ_i and Θ_j are combined into a Gaussian Θ_{ij} by a weighted product of their probability densities:

$$f(x|\Theta_{ij}) = \omega f(x|\Theta_i)^{\alpha_i} f(x|\Theta_j)^{\alpha_j} \tag{1}$$

where α_i and α_j are the combination weights of the Gaussian distributions, $f(x|\Theta)$ is the probability density function (pdf) of a multivariate Gaussian, and ω is some normalization constant. Using this product in a subdivision refinement, the new Gaussian Θ_{ij} will be inserted close to the ridge of the combined probability landscape given by the two Gaussian factors.

Although the formal solution to this non-linear refinement in standard 3D space requires rather complex algebraic computations (multiple matrix-vector products and symmetric matrix inversions), the authors showed that after a bijective mapping into a higher-dimensional space, the refinement of control mesh vertices and

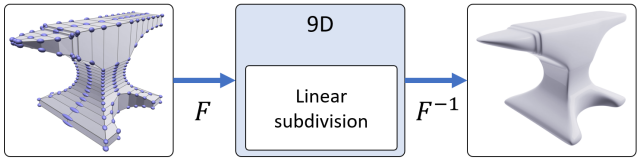


Figure 2: *Visualization of the steps in the GPS algorithm. From left to right: the control mesh is transformed into a 9D space by the use of a bijective map F. There a linear subdivision scheme can be applied. Finally, the mesh is transformed back to standard space, yielding the non-linear subdivision surface.*

their Gaussian covariance information becomes linear. Instead of directly evaluating Equation (1), we can therefore compute the Gaussian-product subdivision by (i) initially transforming the control mesh into this dual space, (ii) applying a standard linear subdivision there, and (iii) finally transforming the limit mesh back to 3D space (Figure 2). Thereby, the computational effort of Gaussian-Product Subdivision is close to the linear scheme, since the only extra computations are the bijective transformations:

1. the initial map $F : (\mu, \Sigma) \in R^3 \times SYM_3 \mapsto q = (\hat{q}, \bar{q}) \in R^9$, with

$$\hat{q} = vech(\Sigma^{-1}) \qquad \bar{q} = \Sigma^{-1}\mu \tag{2}$$

2. the back-transformation $F^{-1} : q \mapsto (\mu, \Sigma)$, where

$$\Sigma = [\hat{q}]^{-1} \qquad \mu = [\hat{q}]^{-1}\bar{q}. \tag{3}$$

Here, $vech(\cdot)$ denotes the half vectorization of the symmetric matrix, linearizing its upper triangular elements, and $[\cdot]$ its inverse operation, restoring the original symmetric matrix.

Finally, it has been shown that the Gaussian-product limit subdivision surface exhibits the same continuity properties as the corresponding linear scheme that is applied in the dual space [PBW19].

2.3. Real-Time Subdivision

First advancements in real-time subdivision used specialized GPU kernels to perform the parallel subdivision in a first pass, and then rendered the subdivided surface to the screen in a second pass [Bun05, SJP05, PEO09]. This, however, requires a large bandwidth for streaming the extensive subdivided mesh data between the GPU cores and global memory. This problem is eliminated by utilizing the tessellation capabilities of modern GPUs. The major challenge for this approach is the direct mapping of tessellated vertices onto the limit subdivision surface. Such a direct evaluation indeed became possible by employing formal derivations of the parametric forms of the limit Catmull-Clark [Sta98b] and Loop subdivision surface [Sta98a] by Jos Stam.

One of the first approaches directly evaluating limit subdivision surface patches in the tessellation unit was introduced by Nießner et al. [NLMD12]. They proposed an adaptive subdivision scheme on the GPU to save computation effort on regular faces. Irregular faces around extraordinary vertices are subdivided to a certain degree using specialized GPU kernels. Between subdivision levels, transition patches are added to avoid T-junctions in the resulting mesh, and patches are finally rendered as bi-cubic B-spline patches. An implementation of their approach is currently also found in Pixar's OpenSubdiv library [PIX21]. The method of Nießner et al. was later also adapted and refined by Brainerd et al. [BFK*16], who added a quad tree storing the information on which patch represents which part of the face. Moreover, the evaluation near extraordinary vertices was simplified, thus eliminating the need for storing explicit transition patches.

In our work, we perform a real-time evaluation and rendering of non-linear Gaussian-Product subdivision surfaces by utilizing the GPU tessellation capabilities similar to Brainerd et al. [BFK*16]. Our pipeline exploits the duality of this probabilistic model to linear subdivision schemes by performing the respective transformations (2) and (3) and evaluating the corresponding 9-dimensional B-spline patches on the fly. Moreover, we compute accurate analytic surface normals of these non-linear surfaces, enabling high-quality rendering and shading. This way, we allow for the first time to directly render high-quality probabilistic Gaussian-Product subdivision surfaces from covariance meshes on the fly, providing real-time feedback for dynamic covariance transformations and interactive manipulations.

In the following, we will first give an overview of our non-linear subdivision evaluation pipeline (Section 3), and then describe the specific details of its realization in an OpenGL pipeline (Section 4 and 5). Finally, Section 6 evaluates the performance and quality of our results.

3. Overview

The major stages of the GPS evaluation pipeline are given in Figure 3. Based on a given input covariance mesh, an initial preprocessing step (Section 4) prepares all necessary vertex buffers and generates a *subdivision plan* on the CPU, comprising additional per-face data structures used for the evaluation of their corresponding GPS surface patches on the GPU later on. This subdivision plan is then streamed to the GPU, where the real-time evaluation is performed in each frame at the Tessellation stage of the rendering pipeline. In the second step, the subdivision plan data is used to compute the control-points that define the bi-cubic B-spline patches corresponding to the limit subdivision surface of the control mesh faces. In the third step, the resulting control points are used to evaluate the B-spline patch at parametric u, v-positions according to the tessellation pattern that generates the patch geometry. The tessellated geometry data is then passed on the pipeline for rasterization. In the following, the individual steps of these stages are described in detail.

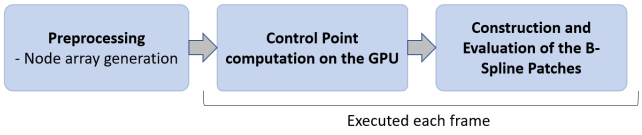


Figure 3: *Real-time tessellation pipeline for evaluating the limit subdivision surface in our approach.*

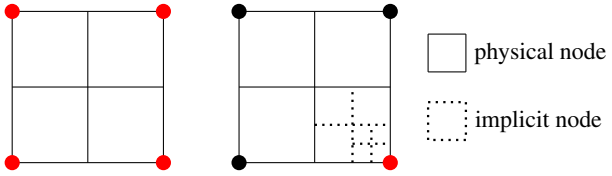


Figure 4: *Left: base control mesh face with 4 extraordinary vertices (red). This gets subdivided into four sub-faces. Right: irregular node with one extraordinary vertex (bottom right).*

4. Mesh Preprocessing

The first step of the pipeline is the computation of a subdivision plan in an offline preprocessing. This plan includes a *nodes array* built for each base mesh face, a *stencil buffer* that contains vertex indices and is referenced by those nodes, and further *mesh data buffers* for storing the covariance data and mesh topology information on the GPU.

4.1. Node Array Generation

The purpose of the per-face node arrays is to subdivide each quad face of the input control mesh into logical evaluation patches that can be effectively tessellated and rasterized by the pipeline. In our implementation each face of the base mesh gets assigned an array of 4 nodes. The order of those nodes determines the position in the face. This splitting ensures that each of the four resulting sub-faces contains at most one extraordinary vertex [BFK*16]. Sub-faces that only contain regular vertices are denoted *regular nodes*, which represent regular B-spline patches that can be directly evaluated. Nodes containing exactly one irregular vertex, are denoted *irregular nodes* (Fig. 4 right). These nodes represent an implicit hierarchy of all occurring sub-faces around an extraordinary vertex. This implicit hierarchy allows an evaluation of the patch at increased granularity as the point of evaluation approaches the extraordinary vertex at a face corner.

4.2. Stencil and Weight Information

Both regular and irregular nodes reference stencils masks of their corresponding control mesh vertices required for the evaluation of their B-spline patches. The stencil vertices of a given node comprise the four vertices of its corresponding face as well as the one-ring neighbor vertices around this face. The indices of these vertices are pushed into a stencil buffer in the preprocessing. For regular nodes, stencil indices are read in the same order in which they are provided to the tessellation unit as control points for the corresponding B-spline patch evaluation. The evaluation of irregular nodes is discussed in more detail in Section 5.

Irregular nodes additionally store a set of weights that are used to compute the hierarchy of subdivision levels. Irregular nodes also reference a set of weights used for the calculation of the control points for each subdivision level that the node represents. These weights can be precomputed constants based on the topology information around an extraordinary vertex of specific valence. In our implementation, we support valences up to 10 and a hierarchy in irregular nodes of depth 10.

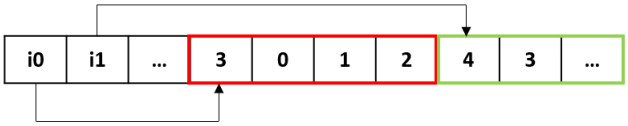


Figure 5: *Schematic layout of set of lists stored in a linear buffer.*

4.3. Vertex and Topology Data Layout

Vertex and covariance data of a covariance mesh are stored in linear GPU buffers (Figure 6). All buffers store their data in (padded) structs of one or multiple 4D vectors to ensure efficient access on the GPU. Vertex positions (Gaussian means) are stored in the μ buffer holding one 4D vector per vertex, while the buffer for covariance data (Σ buffer) uses two 4D vectors to store the 6 independent values of a symmetric covariance matrix. When reading the covariance data, the covariance matrix is unpacked on the GPU into a 3x3 matrix. In order to compute the Gaussian-Product subdivision, the dual-space transformation in Eq. (2) has to be initially computed. The transformed 9D dual-space point of each covariance vertex is stored into 3x4 matrices with their bottom row padded by zero values. The upper triangular part of the remaining submatrix encodes the six values of the transformed covariance matrix $\hat{q}$, while the three values of the lower triangular part store the transformed position vector $\bar{q}$. These 3x4 matrices are stored in the *transformed-vertex* buffer.

To efficiently represent the topology of arbitrary meshes in linear buffers on the GPU, we use a specific buffer layout that compactly stores a set of lists of variable length in a linearly addressed GPU buffer. Examples for such buffers are the *neighbors* buffer, containing the indices of all neighbors of each vertex, and the *linked-faces* buffer, containing the indices of all faces surrounding a vertex. A visualization of the buffer layout employed for this data is given in Figure 5. At the beginning of the buffer an offset lookup table is located. At the index of each vertex, this lookup table holds the offset of the starting position of its neighbors or linked-faces list. At the position given by this offset, the first element stores the length of the list, followed by the list data.

5. Real-Time Evaluation and Rendering

In this section we describe the implementation of the algorithm above in the OpenGL pipeline, and discuss specific details on the buffer accesses. Figure 6 depicts the data flow between shaders. Input buffers (green) are defined or precomputed on the CPU and uploaded to the GPU in the preprocess.

5.1. Tessellation Control

The Tessellation Control Shader (TCS) performs the single Gaussian-Product subdivision step to produce the four sub-nodes of each face. To this end, it first transforms the covariance mesh vertices stored in the input μ buffer and Σ buffer to a corresponding 9-dimensional point q (Eq. (2)). The resulting transformed and subdivided hyperpoints are then stored in the Transformed Vertex buffer for later access. The shader is executed four times for each

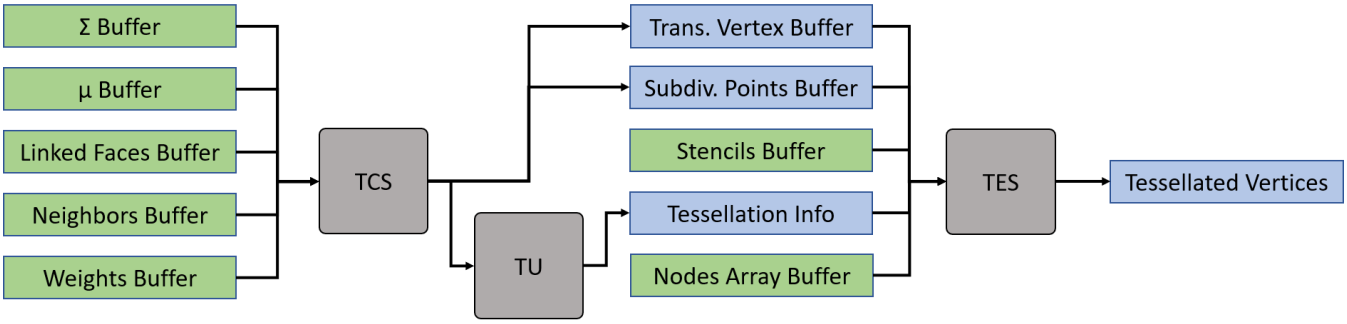


Figure 6: *Flow of data buffers (blue) and input buffers, filled by the CPU (green) over the main processing steps of the OpenGL tessellation stage (gray): Tessellation Control Shader (TCS), Tessellation Unit (TU) and Tessellation Evaluation Shader (TES).*

control mesh face and also accesses the topology information from the linked faces and neighbors buffer.

For irregular nodes bordering extraordinary vertices, the TCS also computes the control points of the different levels of the implicit subdivision hierarchy. To this end, it reads the subdivision weights for each level from the Weights Buffer. The resulting subdivided control points are then written to the Subdivided Points buffer, organized by faces and subdivision depth. These control points are already precomputed in the TCS to enable a quick access in the following Tessellation Evaluation stage. Finally, the Tessellation Unit (TU) is set up for uniform tessellation (equal outer and inner levels).

5.2. Tessellation Evaluation

After the TU, the Tessellation Evaluation Shader (TES) computes the 3D position of each generated vertex. This is done by reading the node responsible for its parametric position from the nodes array. If this node is a regular node, the corresponding 9-dimensional B-spline patch is evaluated and the resulting 9D surface point is transformed back to 3D space. If the node is an irregular node, we first determine the level of the implicit hierarchy the corresponding parametric point maps to (Figure 4 right). Then, the control points corresponding to this level are fetched from the subdivided points buffer. Finally, the B-spline patch is evaluated and the position of the vertex is transformed back into 3D space and passed on down the pipeline.

To make the TES execution more efficient, the array of nodes for each face is read once per face in the TCS and forwarded to the TES threads. The widely used principle of gather reads was also applied to the fetching of subdivided control points of extraordinary nodes. As soon as the required subdivision level is calculated, all control points are fetched and cached inside a local shader array, and then used both for the surface point and surface tangent evaluations.

The surface normal of each tessellated vertex is computed analytically by evaluating the partial derivatives of the 9D B-spline basis functions in the u and v direction. These tangents at the limit hyper surface point are then transformed back to 3D space by employing the analytic derivations for 3D tangents [PBW19]:

$$\begin{aligned} \mu_u &= \Sigma(\bar{q}_u - [\hat{q}_u]\mu) \\ \mu_v &= \Sigma(\bar{q}_v - [\hat{q}_v]\mu) \\ n &= \frac{\mu_u \times \mu_v}{||\mu_u \times \mu_v||} \end{aligned} \tag{4}$$

Here $q_u = (\hat{q}_u, \bar{q}_u)$ denotes the 9D tangent vectors in parametric u direction, μ is the corresponding 3D position of the vertex and μ_u is the 3D tangent vector at position μ in direction u. This is analogous for the v direction.

5.3. Discussion

Our pipeline and data structure setup differs from the previous work of Brainerd et al. [BFK*16] in several aspects. The work of Brainerd et al. uses explicit edge tagging to define semi-sharp features [DKT98], which affects the complexity of the resulting quad hierarchy around these edges. Faces bordering semi-sharp creases require individual crease stencils and need to store explicit crease flags and sharpness values. Additionally, irregular nodes bordering a crease edge need to be decomposed into another, second level of child nodes in a quad hierarchy. In contrast, as the non-linear GPS scheme natively allows to produce features of arbitrary sharpness (Figure 1), we do not need to store any extra tagging information. Moreover, the need for storing hierarchical quad trees for each face is eliminated, and is replaced by a constant set of four sub-nodes per face in our setup.

The parametric evaluation of limit surface patches in our pipeline also requires a transformation of Gaussians to the 9D dual space and back, which however is comparably cheap (Eq. (2) and (3)). The transformed 9D vectors are stored in a GLSL mat3x4 type and all linear combinations are calculated in this matrix form. At the end of the tessellation evaluation stage, the evaluated 9D surface positions are transformed back to 3D space and passed on to the rasterization stage.

6. Results

The performance evaluation of our pipeline as well as the renderings presented in this section were performed on a system with

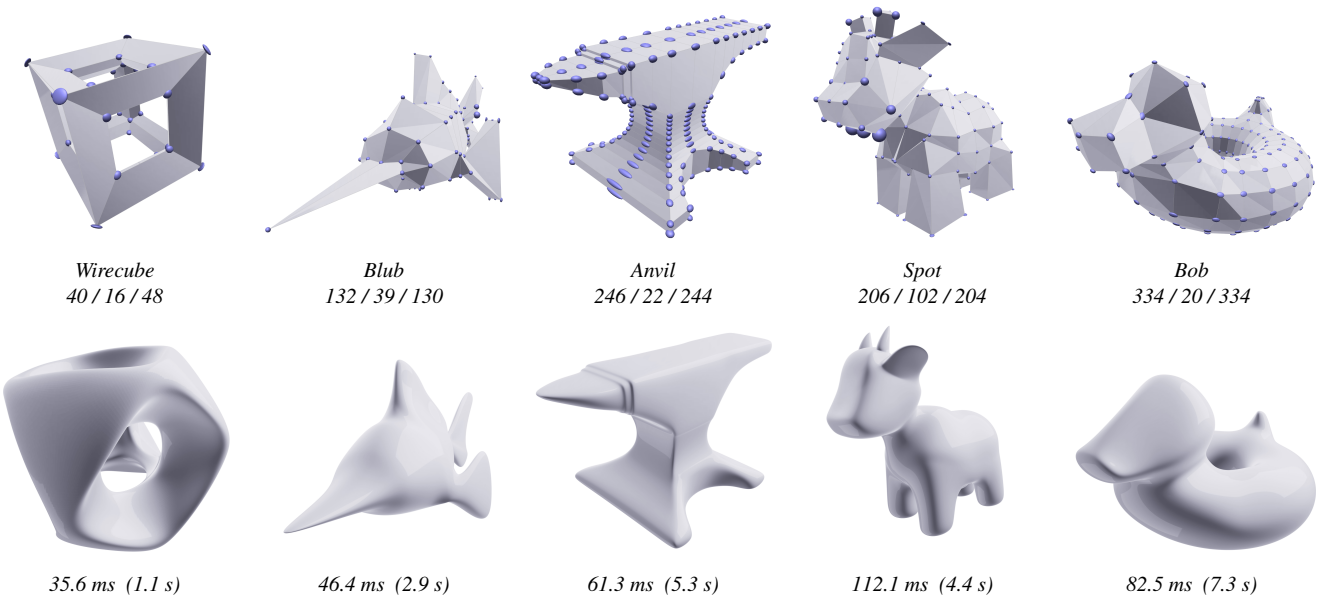


Figure 7: *Information and renders of all covariance meshes used in the performance analysis. Top: input covariance mesh with number of vertices / number of irregular vertices / number of faces. Bottom: rendered GPS limit surface at tessellation level 6, with GPU render time compared to the computation time for CPU subdivision (in parentheses). All images were rendered at 1440p resolution.*

AMD Ryzen 7 1800X CPU, 16GB of RAM and Nvidia GTX 1080 GPU. All meshes were efficiently shaded using material captures in order to avoid extensive bias of the measured performance by the fragment computations.

6.1. Performance Evaluation

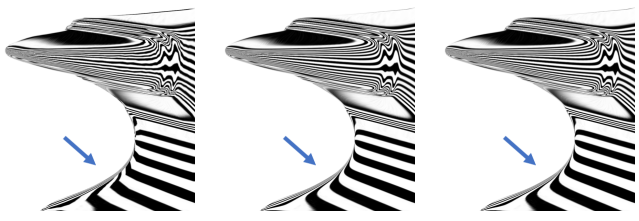
Five meshes were chosen for a time analysis of the pipeline. We picked meshes of varying geometric complexity, Gaussian anisotropy, and different number of vertices. Statistics of the chosen meshes along with their GPS renderings and performance measurements can be seen in Figure 7. On average, we observe speedup factors of 30 – 90 using our GPU subdivision renderer in comparison to a multi-threaded CPU subdivision with 8 threads.

First, performance was compared between the test covariance meshes. As they span a wide variety of vertex and face counts as well as numbers of extraordinary vertices, noticeable differences in the preprocessing and render times are expected. These differences are displayed in Figure 9 at the upper left. An interesting observation can be made, e.g., by comparing the frame render times of the Anvil and the Spot mesh. Although Spot comprises a slightly lower vertex and face count than the Anvil, it requires almost twice the per-frame rendering time. This can be accounted to the much larger number of extraordinary vertices (105 vs. 22 in the anvil mesh) in its topology, leading to a higher number of irregular patches and therefore much more expensive evaluations of patches of increased granularity around the extraordinary vertices.

Next, we investigate the performance characteristics across different tessellation levels. Figure 9 shows the timings for all test meshes between level 1 and 6. All charts exhibit similar performance characteristics that describe an approximate exponential growth over increasing tessellation levels. This is expected, since with increasing tessellation levels the number of output vertices grows exponentially as well. For the meshes used, subdivision levels 1 to 3 consistently exhibit similar frame times on our system, with level 5 marking the start of exploding computational effort (> twice the effort than level 1). In general, it can be seen that for most meshes up to $\sim$ 300 vertices, real-time render times at 60 fps (orange mark) can be achieved up to a tessellation level of 3 to 4.

6.2. Qualitative Evaluation

The primary key for high-quality shading is the availability of appropriate surface normals. The visual quality and continuity of shading using analytic vertex normals was therefore evaluated at different tessellation levels. The quality of the computed normals can be seen in Figure 8. Even at low tessellation levels (up to 2), the reflection lines are smooth to an extend. Increasing the tessellation level, the quality augments further. At the highest tessellation



(a) *L2 Vertex normals* **(b)** *L6 Vertex normals* **(c)** *CPU Reference*

Figure 8: *Reflection lines on the anvil model shaded with vertex normals for different tessellation levels and the CPU reference.*

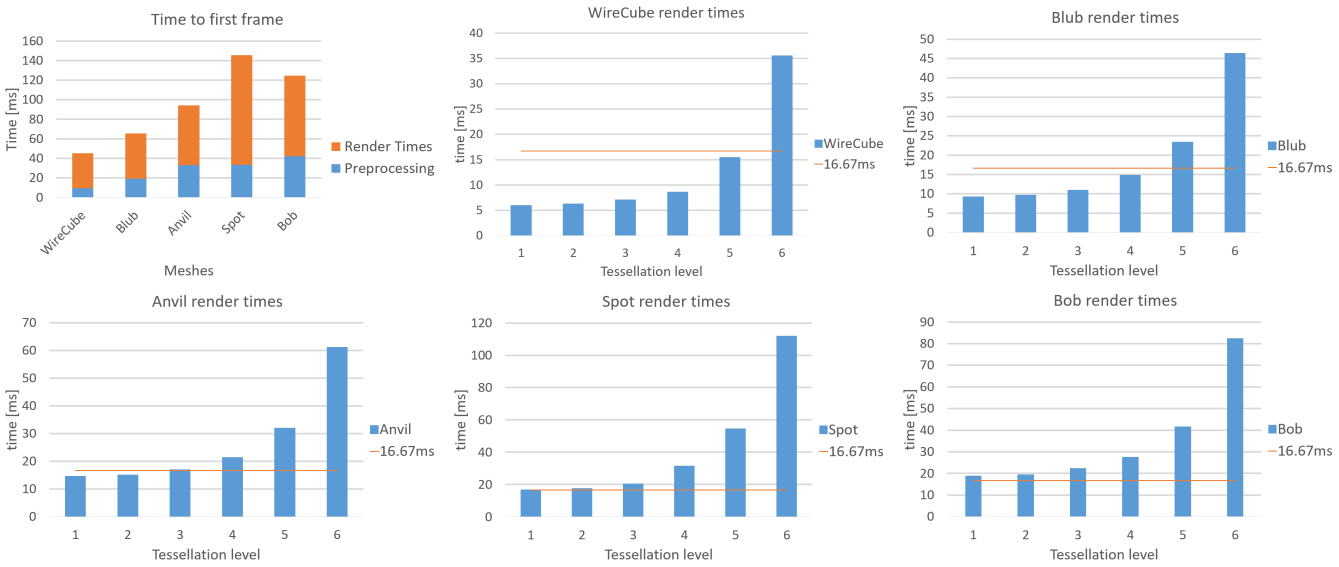


Figure 9: *Render times for all test meshes and all tessellation levels. The orange line in the diagrams indicates the 60 FPS mark. Upper left chart displays a performance comparison between all meshes, both for the preprocessing and the per-frame render time.*

level (6), no visible artifacts can be seen in the reflection lines. The availability of accurate per-vertex normals is vital for use in reflective surface shading, especially when rendering the limit subdivision surfaces at lower, performance-friendly tessellation levels.

Finally, we investigate potential issues of numerical stability and geometric accuracy, which for instance are raised by the fact that the actual subdivision is performed in an intermediate dual space. Transforming the mesh data to and from this space requires several matrix inversions, which are performed only at GPU float precision. In particular, highly anisotropic or flat covariance matrices can be close to being ill-conditioned, and might thus be prone to numerical instabilities under such transformations. To this end, we perform image-based quantitative comparisons between our real-time rendering frames and a comparable CPU-based reference output. For this test, we chose the anvil mesh, which exhibits many highly flat or elongated Gaussians, and the Blub model, exhibiting a strongly scaled-down Gaussian at its tip. This was done by writing the rasterized output world-space positions at each pixel to separate images for each method. The norms of the pixel-based deviation vectors where then measured and normalized by the bounding box diagonal of the model. Figure 10 shows zoomed-in parts of the resulting difference images for two models, which are scaled by a factor $s = 580$ for the Anvil and $s = 530$ for the Blub mesh, and finally inverted (white is zero, black is $1 \times s$). At close inspection, the only measurable deviations are found at individual pixels along the silhouettes of the shapes. These residuals, however, can be observed for any shapes, and are accountable to expectable deviations of the OpenGL output tessellation pattern in the final tessellation level from the output mesh topology of the recursively subdivided reference result.

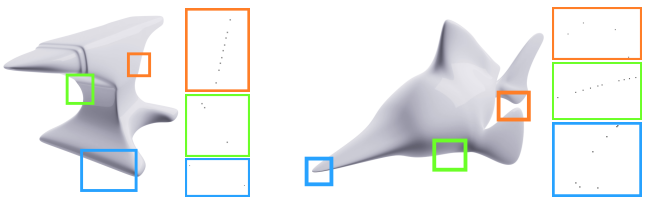


Figure 10: *Renders of two meshes with highlighted regions and their corresponding difference image regions. A white pixel represents no measurable difference between float values. Differences are only measurable along shape silhouettes, which is accounted to different tessellation patterns.*

7. Limitations and Future Work

In this work, we have presented a GPU-based real-time rendering pipeline for probabilistic Gaussian-Product subdivision surfaces. We have shown that the evaluation of this non-linear subdivision scheme can be efficiently performed by exploiting the duality properties of the Gaussian-based schemes to standard linear schemes. The fact that the core subdivision evaluation using bi-cubic B-splines actually operates on a mesh embedded in a 9-dimensional space doesn't pose any further problem to a practical realization on the GPU: once the input position and covariance data of the mesh vertices are transformed using standard matrix-vector arithmetic, the evaluation of the bi-cubic B-spline patches are separable to each dimensional component. Exact analytic normals of the non-linear limit surface can be computed on the fly, when accepting a minimal computational overhead.

One of the major drawbacks of this subdivision scheme at first glance seem to be the increased memory requirements for storing covariance meshes, increasing vertex data storage by factor of 3. In reconstruction scenarios, where the input covariance meshes might

be extracted from Gaussian mixtures that already compress potential huge point cloud data, this apparent overhead seems to be quickly outweighed [PBW19]. However, in modeling applications, where this representation avoids a potential memory overhead for otherwise necessary topological refinement, this limitation might be qualified to some extend. However, comparing the actual memory footprints for defining geometric features in practical modeling examples on both probabilistic and standard subdivision surfaces, will be content of future work.

The efficiency of our pipeline could be improved by introducing different optimizations. Using alternative calculation patterns in the TCS could reduce the number of redundant calculations of control points. Furthermore, memory accesses could be optimized in order to reduce the waiting time on global memory fetches inside shaders.

So far, our technique only works for the probabilistic analogue of Catmull-Clark subdivision applicable to quadrilateral covariance meshes. However, applying it on triangular covariance meshes requires the evaluation of B-spline patches in a similar fashion [Sta01], and only requires the appropriate adaption of the face parametrization and GPU tessellation pattern.

Finally, our evaluation and performance analysis has demonstrated that the generation of smooth limit surfaces around extraordinary vertices is still a critical performance factor, again emphasizing the importance of sanitized high-quality control meshes and skilled quad control mesh modeling.

8. Conclusion

We have presented a thorough technical description of a practical real-time rendering pipeline for Gaussian-Product subdivision surfaces. Our approach bases on previous linear approaches that aim at evaluating the limit surfaces in individual patches by employing the tessellation capabilities on a GPU. We have analyzed quantitative and qualitative aspects of the resulting renderings over different test models and different tessellation levels, and investigated the visual quality and accuracy of the GPU-based subdivision technique as compared to a standard recursive reference solution on the CPU. By enabling real-time feedback in interactive applications based on these enriched mesh model, we envision new development in applications of real-time animation and modeling.

References

[BFK*16] BRAINERD W., FOLEY T., KRAEMER M., MORETON H., NIESSNER M.: Efficient gpu rendering of subdivision surfaces using adaptive quadtrees. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 1–12. doi:10.1145/2897824.2925874. 2, 3, 4, 5

[BLZ00] BIERMANN H., LEVIN A., ZORIN D.: Piecewise smooth subdivision surfaces with normal control. In *Proc. SIGGRAPH* (2000), pp. 113–120. doi:10.1145/344779.344841. 2

[Bun05] BUNNELL M.: Adaptive tessellation of subdivision surfaces with displacement mapping. *GPU Gems 2* (2005), 109–122. 3

[Cas12] CASHMAN T. J.: Beyond catmull–clark? a survey of advances in subdivision surface methods. *Computer Graphics Forum 31*, 1 (2012), 42–61. doi:10.1111/j.1467-8659.2011.02083.x. 2

[CC78] CATMULL E., CLARK J.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-aided design 10*, 6 (1978), 350–355. doi:10.1016/0010-4485(78)90110-0. 1, 2

[DKT98] DEROSE T., KASS M., TRUONG T.: Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), pp. 85–94. doi:10.1145/280814.280826. 2, 5

[DS78] DOO D., SABIN M.: Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design 10*, 6 (1978), 356–360. doi:10.1016/0010-4485(78)90111-2. 2

[Kob00] KOBBELT L.: Square root 3 subdivision. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), pp. 103–112. doi:10.1145/344779.344835. 2

[Loo87] LOOP C.: Smooth subdivision surfaces based on triangles. *Master's thesis, University of Utah, Department of Mathematics* (1987). 2

[NLMD12] NIESSNER M., LOOP C., MEYER M., DEROSE T.: Feature-adaptive gpu rendering of catmull-clark subdivision surfaces. *ACM Trans. Graph. 31*, 1 (Feb. 2012), 6:1–6:11. doi:10.1145/2077341.2077347. 3

[PBW19] PREINER R., BOUBEKEUR T., WIMMER M.: Gaussian-product subdivision surfaces. *ACM Transactions on Graphics 38*, 4 (July 2019), 35:1–35:11. doi:10.1145/3306346.3323026. 2, 3, 5, 8

[PEO09] PATNEY A., EBEIDA M. S., OWENS J. D.: Parallel view-dependent tessellation of catmull-clark subdivision surfaces. In *Proceedings of the conference on high performance graphics 2009* (2009), pp. 99–108. doi:10.1145/1572769.1572785. 3

[PIX21] PIXAR: Opensubdiv api (version 3.4.4), 2021. URL: https://graphics.pixar.com/opensubdiv/. 3

[SJP05] SHIUE L.-J., JONES I., PETERS J.: A realtime gpu subdivision kernel. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 1010–1015. doi:10.1145/1186822.1073304. 3

[Sta98a] STAM J.: Evaluation of loop subdivision surfaces. In *SIGGRAPH'98 CDROM Proceedings* (1998). 3

[Sta98b] STAM J.: Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *Siggraph* (1998), vol. 98, pp. 395–404. doi:10.1145/280814.280945. 2, 3

[Sta01] STAM J.: On subdivision schemes generalizing uniform b-spline surfaces of arbitrary degree. *Computer Aided Geometric Design 18*, 5 (2001), 383–396. doi:10.1016/S0167-8396(01)00038-3. 8